Title:	Linearity and Group Homomorphism Testing / Testing Hadamard Codes
Name:	Sofya Raskhodnikova[1], Ronitt Rubinfeld[2]
Affil./Addr. 1:	Pennsylvania State University
Affil./Addr. 2:	MIT and Tel Aviv University
Keywords:	Property testing, sublinear-time algorithms, linearity of functions, group homomorphism, error-correcting codes
SumOriWork:	1993; Blum, Luby, Rubinfeld

Linearity and Group Homomorphism Testing / Testing Hadamard Codes

Sofya Raskhodnikova[1], Ronitt Rubinfeld[2]

[1] Pennsylvania State University
[2] MIT and Tel Aviv University

Years aud Authors of Summarized Original Work

1993; Blum, Luby, Rubinfeld

Keywords

Property testing, sublinear-time algorithms, linearity of functions, group homomorphism, error-correcting codes

Problem Definition

In this article, we discuss the problem of testing linearity of functions and, more generally, testing whether a given function is a group homomorphism. An algorithm for this problem, given by Blum, Luby, and Rubinfeld [9], is part of or is a special case of many important property testers for algebraic properties. Originally designed for program checkers and self-correctors, it has found uses in Probabilistically Checkable Proofs (PCPs), which are an essential tool in proving hardness of approximation.

We start by formulating an important special case of the problem, testing linearity of Boolean functions. A function $f : \{0,1\}^n \to \{0,1\}$ is *linear* if for some $a_1, a_2, \ldots, a_n \in \{0,1\}$,

$$f(x_1, x_2, \ldots, x_n) = a_1 x_1 + a_2 x_2 + \cdots a_n x_n.$$

The operations in this definition are over $\mathbb{F}_2$. That is, given vectors $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$, where $x_1, \ldots, x_n, y_1, \ldots, y_n \in \{0,1\}$, the vector $\mathbf{x} + \mathbf{y} = (x_1 + y_1 \bmod 2, \ldots, x_n + y_n \bmod 2)$. There is another, equivalent, definition of linearity of Boolean functions over $\{0,1\}^n$: a function f is *linear* if for all $x, y \in \{0,1\}^n$,

$$f(x) + f(y) = f(x + y).$$

A generalization of a linear function, defined above, is a group homomorphism. Given two finite groups, $(G, \circ)$ and $(H, \star)$, a *group homomorphism* from G to H is a function $f : G \to H$ such that for all elements $x, y \in G$,

$$f(x) \star f(y) = f(x \circ y).$$

We would like to test (approximately) whether a given function is linear or, more generally, is a group homomorphism. Next, we define the property testing framework [23; 12]. Linearity testing was the first problem studied in this framework. The linearity tester of Blum, Luby, and Rubinfeld [9] actually preceded the definition of this framework and served as an inspiration for it. Given a proximity parameter $\epsilon \in (0, 1)$, a function is ϵ-far from satisfying a specific property $\mathcal{P}$ (such as being linear or being a group homomorphism) if it has to be modified on at least an ϵ fraction of its domain in order to satisfy $\mathcal{P}$. A function is ϵ-close to $\mathcal{P}$ if it is not ϵ-far from it. A tester for property $\mathcal{P}$ gets a parameter $\epsilon \in (0, 1)$ and an oracle access to a function f. It must accept with probability[3] at least 2/3 if the function f satisfies property $\mathcal{P}$ and reject with probability at least 2/3 if f is ϵ-far from satisfying $\mathcal{P}$. Our goal is to design an efficient tester for group homomorphism.

Alternative formulation Another way of viewing the same problem is in terms of error correcting codes. Given a function $f : G \to H$, we can form a codeword corresponding to f by listing the values of f on all points in the domain. The *homomorphism* code is the set of all codewords that correspond to homomorphisms from G to H. This is an error-correcting code with large distance because, for two different homomorphisms $f, g : G \to H$, the fraction of points $x \in G$ on which $f(x) = g(x)$ is at most 1/2. In the special case when G is $\{0, 1\}^n$ and H is $\{0, 1\}$, we get the Hadamard code. Our goal can be formulated as follows: Design an efficient algorithm that tests whether a given string is a codeword of a homomorphism code (or ϵ-far from it).

Key Results

The linearity (homomorphism) tester designed by Blum, Luby, and Rubinfeld [9] repeats the following test several times, until the desired success probability is reached, and accepts iff all iterations accept.

Algorithm 1: BLR Linearity (Homomorphism) Test

input : Oracle access to an unknown function $f : G \to H$.

1. Pick $x, y \in G$ uniformly and independently at random.
2. Query f on x, y, and $x + y$ to find out $f(x), f(y)$, and $f(x + y)$.
3. **Accept** if $f(x) + f(y) = f(x + y)$; otherwise, **reject**.

Blum et al. [9] and Ben-Or et al. [7] showed that $O(1/\epsilon)$ iterations of the BLR test suffice to get a property tester for group homomorphism. (The analysis in [9] worked for a special case of the problem, and [7] extended it all groups). It is not hard to prove that $\Omega(1/\epsilon)$ queries are required to test for linearity and, in fact, any

[3] The choice of error probability in the definition of the tester is arbitrary. Using standard techniques, a tester with error probability 1/3 can be turned into a tester with error probability $\delta \in (0, 1/3)$ by repeating the original tester $O(\log \frac{1}{\delta})$ times and taking the majority answer.

non-trivial property, so the resulting tester is optimal in terms of the query complexity and the running time.

Lots of effort went into understanding the rejection probability of the BLR test for functions that are ϵ-far from homomorphisms over various groups and, especially, for the case $F = \{0,1\}^n$ (see [17] and references therein). A nice exposition of the analysis for the latter special case, which follows the Fourier-analytic approach of Bellare et al. [5], can by found in the book by O'Donnell [21].

Several works [26; 24; 14; 8; 25] showed how to reduce the number of random bits required by homomorphism tests. In the natural implementation of the BLR test, $2\log|G|$ random bits per iteration are used to pick x and y. Shpilka and Wigderson [25] gave a homomorphism test for general groups that needs only $(1+o(1))\log_2|G|$ random bits.

The case when G is a subset of an infinite group, f is a real-valued function, and the oracle query to f returns a finite-precision approximation to $f(x)$ has been considered in [11; 2; 10; 19; 20]. These works gave testers with query complexity independent of the domain size (see [18] for a survey).

Applications

Self-testing/correcting programs The linearity testing problem was motivated in [9] by applications to self-testing and self-correcting of programs. Suppose you are given a program that is known to be correct on most inputs, but has not been checked (or, perhaps, is even known to be incorrect) on remaining inputs. A *self-tester* for f is an algorithm that can quickly verify whether a given program that supposedly computes f is correct on most inputs, without the aid of another program for f that has already been verified. A *self-corrector* for f is an algorithm that takes a program that correctly computes f on most inputs and uses it to correctly compute f on all inputs.

Blum et al. [9] used their linearity test to construct self-testers for programs intended to compute various homomorphisms. Such functions include integer, polynomial, matrix and modular multiplication and division. Once it is verified that a program agrees on most inputs with some homomorphism, the task of determining whether it agrees with the *correct* homomorphism on most inputs becomes much easier.

For programs intended to compute homomorphisms, it is easy to construct self-correctors: Suppose a program outputs $f(x)$ on input x, where f agrees on most inputs with a homomorphism g. Fix a constant c. Consider the algorithm that, on input x, picks $c\log 1/\delta$ values y from the domain G uniformly at random, computes $f(x+y)-f(y)$, and outputs the value that is seen most often, breaking ties arbitrarily. If f is $\frac{1}{8}$-close to g then, since both y and $x+y$ are uniformly distributed in G, it is the case that for at least $3/4$ of the choices of y, both $g(x+y)=f(x+y)$ and $g(y)=f(y)$, in which case $f(x+y)-f(y)=g(x)$. Thus, it is easy to show that there is a constant c such that if f is $\frac{1}{8}$-close to a homomorphism g, then for all x, the above algorithm outputs $g(x)$ with probability at least $1-\delta$.

Probabilistically Checkable Proofs We discussed an equivalent formulation of the linearity testing problem in terms of testing whether a given string is a codeword of a Hadamard code. This formulation has been used in proofs of hardness of approximation of some NP-hard problems and to construct PCP systems that can be verified with a few queries (see, e.g., [3; 13]).

The BLR Test as a Building Block The BLR test has been generalized and extended in many ways, as well as used as a building block in other testers. One

generalization, particularly useful in PCP constructions, is to testing if a given function is a polynomial of low degree (see, e.g., [16; 1; 15]). Other generalizations include tests for long codes [6; 13] and tests of linear consistency among multiple functions [4]. An example of an algorithm that uses the BLR test as a building block is a tester by Parnas, Ron, and Samorodnitsky [22] for the *singleton* property of functions $f : \{0,1\}^n \to \{0,1\}$, namely, the property that the function $f(x) = x_i$ for some $i \in [1,n]$.

Open Problems

We discussed that the BLR test can be used to check whether a given string is a Hadamard codeword or far from it. For which other codes can such a check be performed efficiently? In other words, which codes are locally testable? We refer the reader to the entry "Locally Testable Codes".

Which other properties of functions can be efficiently tested in the property testing model? Some examples are given in the entries "Testing Juntas and Related Properties of Boolean Functions" and "Monotonicity Testing". Testing properties of graphs is discussed in the entries "Testing Bipartiteness in the Dense-Graphs Model" and "Testing Bipartiteness of Graphs in Sublinear Time".

Cross-References

Testing Juntas and Related Properties of Boolean Functions
Locally Testable Codes
Monotonicity Testing
Testing if an Array Is Sorted
Testing Bipartiteness in the Dense-Graphs Model
Testing Bipartiteness of Graphs in Sublinear Time
Learning Heavy Fourier Coefficients of Boolean Functions
Error correction

Acknowledgements

The first author was supported in part by NSF award CCF-1422975 and by NSF CAREER award CCF-0845701.

Recommended Reading

1. Alon N, Kaufman T, Krivilevich M, Litsyn S, Ron D (2003) Testing low-degree polynomials over GF(2). In: Proceedings of RANDOM '03, pp 188–199
2. Ar S, Blum M, Codenotti B, Gemmell P (2003) Checking approximate computations over the reals. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing, pp 786–795
3. Arora S, Lund C, Motwani R, Sudan M, Szegedy M (1998) Proof verification and the hardness of approximation problems. J ACM 45(3):501–555
4. Aumann Y, Håstad J, Rabin MO, Sudan M (2001) Linear-consistency testing. J Comput Syst Sci 62(4):589–607
5. Bellare M, Coppersmith D, Håstad J, Kiwi M, Sudan M (1996) Linearity testing over characteristic two. IEEE Transactions on Information Theory 42(6):1781–1795
6. Bellare M, Goldreich O, Sudan M (1998) Free bits, PCPs, and nonapproximability—towards tight results. SIAM J Comput 27(3):804–915

7. Ben-Or M, Coppersmith D, Luby M, Rubinfeld R (2008) Non-Abelian homomorphism testing, and distributions close to their self-convolutions. Random Struct Algorithms 32(1):49–70
8. Ben-Sasson E, Sudan M, Vadhan S, Wigderson A (2003) Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on the Theory of Computing, pp 612–621
9. Blum M, Luby M, Rubinfeld R (1993) Self-testing/correcting with applications to numerical problems. JCSS 47:549–595
10. Ergun F, Kumar R, Rubinfeld R (2001) Checking approximate computations of polynomials and functional equations. SIAM J Comput 31(2):550–576
11. Gemmell P, Lipton R, Rubinfeld R, Sudan M, Wigderson A (1991) Self-testing/correcting for polynomials and for approximate functions. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, pp 32–42
12. Goldreich O, Goldwasser S, Ron D (1998) Property testing and its connection to learning and approximation. J ACM 45(4):653–750
13. Håstad J (2001) Some optimal inapproximability results. J ACM 48(4):798–859
14. Hastad J, Wigderson A (2003) Simple analysis of graph tests for linearity and PCP. Random Structures and Algorithms 22(2):139–160
15. Jutla CS, Patthak AC, Rudra A, Zuckerman D (2009) Testing low-degree polynomials over prime fields. Random Struct Algorithms 35(2):163–193
16. Kaufman T, Ron D (2006) Testing polynomials over general fields. SIAM J Comput 36(3):779–802
17. Kaufman T, Litsyn S, Xie N (2010) Breaking the epsilon-soundness bound of the linearity test over GF(2). SIAM J Comput 39(5):1988–2003
18. Kiwi M, Magniez F, Santha M (2001) Exact and approximate testing/correcting of algebraic functions: A survey. Electronic Colloqium on Computational Complexity 8(14)
19. Kiwi M, Magniez F, Santha M (2003) Approximate testing with error relative to input size. JCSS 66(2):371–392
20. Magniez F (2005) Multi-linearity self-testing with relative error. Theory Comput Syst 38(5):573–591
21. O'Donnell R (2014) Analysis of Boolean Functions. Cambridge University Press
22. Parnas M, Ron D, Samorodnitsky A (2002) Testing basic Boolean formulae. SIAM J Discrete Math 16(1):20–46
23. Rubinfeld R, Sudan M (1996) Robust characterizations of polynomials with applications to program testing. SIAM J Comput 25(2):252–271
24. Samorodnitsky A, Trevisan L (2000) A PCP characterization of NP with optimal amortized query complexity. In: stoc00, pp 191–199
25. Shpilka A, Wigderson A (2006) Derandomizing homomorphism testing in general groups. SIAM J Comput 36(4):1215–1230
26. Trevisan L (1998) Recycling queries in PCPs and in linearity tests. In: Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, pp 299–308